Computer Graphics Lab 4 Report

一、实验内容

隐藏面消除的画家算法

隐藏面消除的朴素 Z-Buffer 算法

二、测试环境

OS: OpenSUSE Leap 42.2

CPU: Intel® Core™ i7-6500U @ 2.60GHz

RAM: 8GB DDR4 2133MHz

Programming Language: GCC-C++ 6.3.0

三、操作说明

程序由固定格式的输入绘制多边形

四、实现过程画家算法：

定义数据结构：

颜色(Color)

屏幕(Scene)

二维向量(Point2D)

三维向量(Point3D)

边(Edge)

二维多边形(Polygon2D)

三维多边形(Polygon3D)

这里仅展示 Polygon3D 的结构

```
class Polygon3D {
        public:
                int vertexCount, nMin, nMax, uMin, uMax, vMin, vMax;

                Color color;

                Point3D *vertexSequence;

                Polygon3D() {
                        vertexCount = 0;

                        nMin = 0x7FFFFFFF;
                        nMax = 0x80000000;

                        uMin = 0x7FFFFFFF;
                        uMax = 0x80000000;

                        vMin = 0x7FFFFFFF;
                        vMax = 0x80000000;
                }
```

对所有三维多边形按 远点深度（nMin）由小到大排序。

从排序的最小三维多边形开始执行画家算法：

对比当前三维多边形大的每个三维多边形依次判断两者顺序是否正确，有五个判据按顺序运行，只要有一个成立即可：

1. 后者的 nMax 大于前者的 nMin
2. 两者包围盒无交

通过预先计算两个三维多边形的 u, v 的最小和最大值，只要前者的某个最大值小于后者的 最小值或者前者的某个最小值大于后者的最大值即可。

3. 后者所有顶点在前者的可见一侧

三维空间下对前者的每个顶点和后者的前三个定点构成三个三维矢量，求出三个矢量的混合积(有向体积)，判断其有向体积是否大于 0，从而判断是否在可见一侧。

4. 前者所有顶点在后者的不可见一侧 判断方法与（3）相同。
5. 两个三维多边形投影无交，或两个三维多边形形投影有交，且所交区域内一点对应前者的 n 小于后者。

取平面内所有点，由射线法判断是否在两个二维多边形(三维多边形的投影)内，若有说明两者有交，判断该交点深度是否正确即可，若没有说明无交，返回 true。

如果所有判据为假，则交换两者在数组中的位置，重新从头判断。

如果只剩余一个三角形，算法结束。

```
void ReSort(Polygon3D * begin, Polygon3D * end) {
	if (end - begin == 1)
		return;

	sort(begin, end);

beforeLoop:
	for (auto ptr = begin + 1; ptr != end; ptr++)
		if (ptr -> nMin < begin -> nMax) {
			if (begin -> SimpleInterSect(*ptr) == false)
				continue;

			if (ptr -> Locate(*begin) == 1)
				continue;

			if (begin -> Locate(*ptr) == -1)
				continue;

			if (begin -> InterSect(*ptr) != 1)
				continue;
			else {
				swap(*begin, *ptr);
				goto beforeLoop;
			}
		}

	sort(begin + 1, end);
}
```

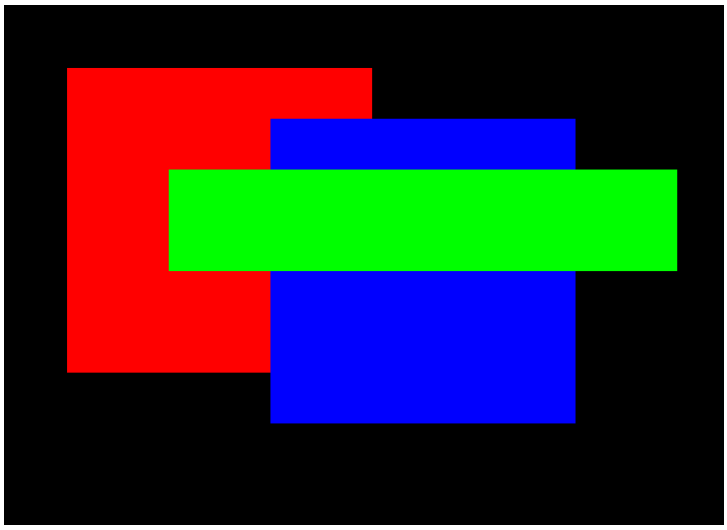
朴素的 **Z-Buffer** 算法：

对每个像素，初始 n 值设为最小，保存当前的最大 z 值和对应的颜色值，依次对每个像素，每个三维多边形，如果像素在投影的二维多边形内部，求得当前的深度，与 Z-buffer 中的深度比较和替换， 终每个像素的颜色是深度最大的颜色。

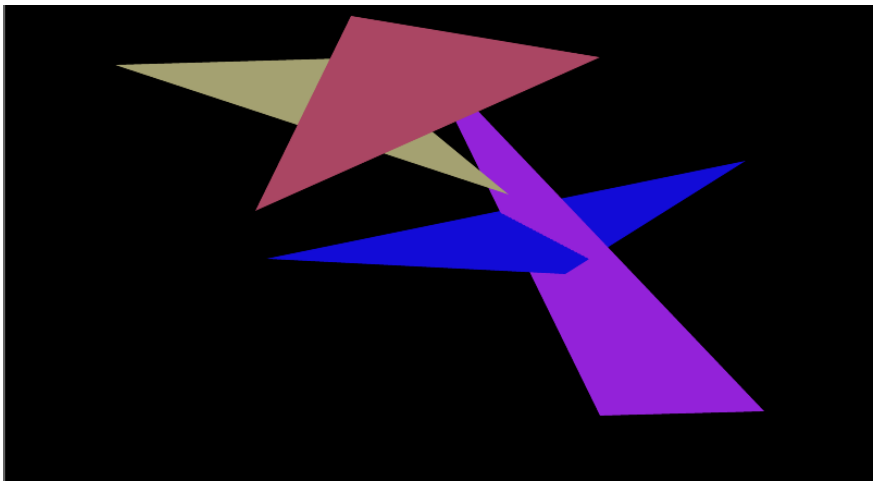
```
void Z_Buffer_Algorithm() {
    for (int p = 0; p < polygonCount; p++)
        for (int i = 0; i < maxY; i++)
            for (int k = 0; k < 5; k++) {
                double yCurr = i + 0.1 + k * 0.2;

                for (int j = 0; j < maxX; j++)
                    for (int l = 0; l < 5; l++) {
                        double xCurr = j + 0.1 + l * 0.2;

                        Point2D currPoint(xCurr, yCurr);

                        if (polygon2D[p].Include(currPoint)) {
                            double nCurr = polygon3D[p].getN(currPoint);

                            if (nCurr > zBuffer[j * 5 + l][i * 5 + k]) {
                                zBuffer[j * 5 + l][i * 5 + k] = nCurr;
                                color[j * 5 + l][i * 5 + k] = polygon3D[p].color;
                            }

                        }
                    }
            }

    for (int i = 0; i < maxX; i++)
        for (int j = 0; j < maxY; j++) {
            Color currColor;

            for (int k = 0; k < 5; k++)
                for (int l = 0; l < 5; l++)
                    currColor = currColor + color[i * 5 + k][j * 5 + l] * val[k][l];
            scene.color[i][j] = currColor / SUM;

            scene.color[i][j].red = round(scene.color[i][j].red);
            scene.color[i][j].green = round(scene.color[i][j].green);
            scene.color[i][j].blue = round(scene.color[i][j].blue);
        }
}
```

五、结果示例

画家算法：



朴素的 Z-Buffer 算法：



六、小结

本实验通过画家算法和 Z-Buffer 算法分别实现了隐藏面消除，其中画家算法速度较快，但不能绘制三角形交叉和循环遮挡的情况，朴素的 Z-Buffer 速度较慢，但能够实现任意三维多边形的绘制，还可以通过扫描线优化。